

CPU Scheduling



CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. In this chapter, we introduce the basic scheduling concepts and discuss in great length CPU scheduling. FCFS, SJF, Round-Robin, Priority, and the other scheduling algorithms should be familiar to the students. This is their first exposure to the idea of resource allocation and scheduling, so it is important that they understand how it is done. Gantt charts, simulations, and play acting are valuable ways to get the ideas across. Show how the ideas are used in other situations (like waiting in line at a post office, a waiter time sharing between customers, even classes being an interleaved Round-Robin scheduling of professors).

A simple project is to write several different CPU schedulers and compare their performance by simulation. The source of CPU and I/O bursts may be generated by random number generators or by a trace tape. The instructor can make the trace tape up in advance to provide the same data for all students. The file that I used was a set of jobs, each job being a variable number of alternating CPU and I/O bursts. The first line of a job was the word JOB and the job number. An alternating sequence of CPU n and I/O n lines followed, each specifying a burst time. The job was terminated by an END line with the job number again. Compare the time to process a set of jobs using FCFS, Shortest-Burst-Time, and Round-Robin scheduling. Round-Robin is more difficult, since it requires putting unfinished requests back in the ready queue.

Exercises

- 5.1 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

Answer: I/O-bound programs have the property of performing only a small amount of computation before performing IO. Such programs typically do not use up their entire CPU quantum. CPU-bound programs,

on the other hand, use their entire quantum without performing any blocking IO operations. Consequently, one could make better use of the computer's resources by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

- 5.2 Discuss how the following pairs of scheduling criteria conflict in certain settings.
- CPU utilization and response time
 - Average turnaround time and maximum waiting time
 - I/O device utilization and CPU utilization

Answer:

- CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.
- Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could however starve long-running tasks and thereby increase their waiting time.
- I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

- 5.3 Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?
- $\alpha = 0$ and $\tau_0 = 100\text{milliseconds}$
 - $\alpha = 0.99$ and $\tau_0 = 10\text{milliseconds}$

Answer: When $\alpha = 0$ and $\tau_0 = 100\text{milliseconds}$, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10\text{milliseconds}$, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memory-less, and simply predicts the length of the previous burst for the next quantum of CPU execution.

- 5.4 Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5 , all at time 0.

- Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
- What is the turnaround time of each process for each of the scheduling algorithms in part a?
- What is the waiting time of each process for each of the scheduling algorithms in part a?
- Which of the schedules in part a results in the minimal average waiting time (over all processes)?

Answer:

- The four Gantt charts are
- Turnaround time

	FCFS	RR	SJF	Priority
P_1	10	19	19	16
P_2	11	2	1	1
P_3	13	7	4	18
P_4	14	4	2	19
P_5	19	14	9	6

- Waiting time (turnaround time minus burst time)

	FCFS	RR	SJF	Priority
P_1	0	9	9	6
P_2	10	1	0	0
P_3	11	5	2	16
P_4	13	3	1	18
P_5	14	9	4	1

- Shortest Job First

5.5 Which of the following scheduling algorithms could result in starvation?

- First-come, first-served
- Shortest job first
- Round robin

d. Priority

Answer: Shortest job first and priority-based scheduling algorithms could result in starvation.

5.6 Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

- a. What would be the effect of putting two pointers to the same process in the ready queue?
- b. What would be the major advantages and disadvantages of this scheme?
- c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer:

- a. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.
- b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.
- c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quanta possible in the Round-Robin scheme.

5.7 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

- a. The time quantum is 1 millisecond
- b. The time quantum is 10 milliseconds

Answer:

- The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of $1/1.1 * 100 = 91\%$.
- The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore $10 * 1.1 + 10.1$ (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore $20/21.1 * 100 = 94\%$.

- 5.8 Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

Answer: The program could maximize the CPU time allocated to it by not fully utilizing its time quantum. It could use a large fraction of its assigned quantum, but relinquish the CPU before the end of the quantum, thereby increasing the priority associated with the process.

- 5.9 Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α ; when it is running, its priority changes at a rate β . All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

- What is the algorithm that results from $\beta > \alpha > 0$?
- What is the algorithm that results from $\alpha < \beta < 0$?

Answer:

- FCFS
- LIFO

- 5.10 Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

- FCFS
- RR
- Multilevel feedback queues

Answer:

- FCFS—discriminates against short jobs since any short jobs arriving after long jobs will have a longer waiting time.
- RR—treats all jobs equally (giving them equal bursts of CPU time) so short jobs will be able to leave the system faster since they will finish first.
- Multilevel feedback queues—work similar to the RR algorithm—they discriminate favorably toward short jobs.

- 5.11 Using the Windows XP scheduling algorithm, what is the numeric priority of a thread for the following scenarios?

- A thread in the `REALTIME_PRIORITY_CLASS` with a relative priority of `HIGHEST`.
- A thread in the `NORMAL_PRIORITY_CLASS` with a relative priority of `NORMAL`.
- A thread in the `HIGH_PRIORITY_CLASS` with a relative priority of `ABOVE_NORMAL`.

Answer:

- a. 26
- b. 8
- c. 14

5.12 Consider the scheduling algorithm in the Solaris operating system for time sharing threads:

- a. What is the time quantum (in milliseconds) for a thread with priority 10? With priority 55?
- b. Assume a thread with priority 35 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?
- c. Assume a thread with priority 35 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?

Answer:

- a. 160 and 40
- b. 35
- c. 54

5.13 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

$$\text{Priority} = (\text{Recent CPU usage} / 2) + \text{Base}$$

where base = 60 and *recent CPU usage* refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P_1 is 40, process P_2 is 18, and process P_3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Answer: The priorities assigned to the processes are 80, 69, and 65 respectively. The scheduler lowers the relative priority of CPU-bound processes.